Science of Computer Programming 9 (1987) 107-136
North-Holland

A HOARE-LIKE PROOF SYSTEM FOR ANALYSING THE COMPUTATION TIME OF PROGRAMS

Hanne Riis NIELSON
Institute of Electronic Systems, Aalborg University Centre, DK-9000 Aalborg, Denmark

Communicated by K. Apt
Received January 1985
Revised January 1987

Abstract. Versions of Hoare logic have been introduced to prove partial and total correctness properties of programs. In this paper it is shown how a Hoare-like proof system for while programs may be extended to prove properties of the computation time as well. It should be stressed that the system does *not* require the programs to be modified by inserting explicit operations upon a clock variable. We generalize the notions of arithmetically sound and complete and show that the proof system satisfies these. Also we derive formal rules corresponding to the informal rules for determining the computation time of while programs. The applicability of the proof system is illustrated by an example, the bubble sorting algorithm.

1. Introduction

The motivation for this work is the similarity between a correctness property of a program and a property of its worst-case computation time. In both cases we have a performance guarantee: The correctness property states how the output of a computation is related to the input and a property of the worst-case computation time gives an upper bound on the computation time (expressed in terms of the input). Hoare-like proof systems have turned out to be successful for proving programs correct. In this paper we shall show how they can be extended to prove properties of the computation time as well.

Several techniques have been developed for analysing the worst-case computation time of programs. A large number of programs have been analysed using these techniques in, e.g., [2]. We want to obtain a (arithmetically sound and complete) formal proof system for computation time that allows us to view these informal analyses as informal proofs in the formal proof system.

The design of the proof system has been motivated by the *informal rules* for analysing the computation time of while programs suggested by [2]. The rules are:
- *assignment*: the computation time is $\mathcal{O}(1)$, that is, it is bounded by a constant;
- *composition*: the computation time is, to within a constant factor, the largest of the computation times of the two statements;

- *iteration*: the computation time of the loop is the sum, over all the times round the loop, of the time to execute the body and the time for evaluating the condition (usually $\mathcal{O}(1)$); often this time is, neglecting constant factors, the product of the number of times the loop is executed and the maximal computation time for the body of the loop.

An important property of these informal rules is that they give rise to analyses that reflect the *structure* of the programs: Given properties of the computation times for the constituents of a composite program the rules specify a property of the computation time of the complete program. The formal proof system will reflect this.

It should be stressed that the proof system will *not* require the programs to be modified by inserting explicit operations upon a clock variable [9, 16]. In this approach one proves a property of the computation time of a program by first inserting statements updating a clock variable (that records the time used so far) and then one uses a traditional proof system for, e.g., total correctness. We claim that this approach does not allow natural formalisations of existing informal analyses of the computation time because the program transformation does not have a counterpart in the traditional informal analyses. It is also worth noticing that the clock variable is treated as an ordinary program variable in the formal proof. This implies that there are no disciplines ensuring that a property of the computation time of a program does not involve final values of the program variables. (As stated above one wants the computation time expressed in terms of the initial values of the variables in the program.)

To illustrate the essence of the present proof system let us consider a proof rule for the composite construct $c_1;c_2$. For the sake of simplicity assume that we have given properties $\mathbf{time}_i \leq T_i$ holding for the computation time of c_i $(i=1,2)$. Here the $\mathbf{time}_i$ are special variables in the assertion language denoting the (unknown) computation times of the programs c_i. The T_i are terms in the assertion language and it is assumed that the free variables of T_i are program variables occurring in c_i. The properties $\mathbf{time}_i \leq T_i$ express that the value of T_i in the initial state of c_i will be an upper bound on the computation time of c_i.

The computation time of $c_1;c_2$ will be $\mathbf{time}_1+\mathbf{time}_2$ but we cannot expect T_1+T_2 to be an upper bound for the simple reason that the references of T_2 do not refer to the values of the variables *before* executing c_1 but rather to the values *after* c_1 has been executed. In many cases c_1 might be such that the variables occurring freely in c_2 are not changed by c_1 (and then $\mathbf{time}_1+\mathbf{time}_2 \leq T_1+T_2$ will indeed hold for the computation time of $c_1;c_2$), but this is not the case in general. A solution to the problem is to keep track of how the values of the variables are modified by c_1. This information may then be used to transform the term T_2 into another term T_2' with the property that its value in the initial state of c_1 never will be less than the value of T_2 in the final state of c_1 (which is the initial state of c_2). Then $\mathbf{time}_1+\mathbf{time}_2 \leq T_1+T_2'$ is guaranteed to hold for the computation time of $c_1;c_2$.

The traditional Hoare-like proof systems (see e.g. [3]) can easily be used to prove relations between the initial and the final values of the variables by taking a so-called

snapshot of the initial values in the pre-condition. Manna and Pnueli [10] have suggested a proof system where the post-conditions always express *relations* between the initial and the final values rather than just properties of their final values. That work has been further developed by Jones [8] who argues that one quite often is interested in proving properties about the input/output behaviour of programs and that it therefore is convenient to have a proof system that directly supports such relationships in the post-conditions. The proof system for computation time to be presented here will be based upon an appropriate version of such a proof system but the development could as well have been performed for the more traditional proof systems of [3].

In Section 2 we introduce the language of while programs. The semantics and computation time is defined by an operational semantics and is based upon a notion of computational model. In Section 3 we present the proof system $\boldsymbol{T}$ for total correctness and extend it to the proof system $\boldsymbol{R}$ for proving properties of the computation time. Soundness and completeness results for $\boldsymbol{R}$ (and $\boldsymbol{T}$) are considered in Section 4. In Section 5 we discuss the applicability of the proof system. It is shown how formalised versions of the informal rules mentioned earlier can be derived in $\boldsymbol{R}$ and how the well-known analysis of the bubble sorting algorithm can be formalised in $\boldsymbol{R}$. Finally, Section 6 contains the concluding remarks.

2. The language of while programs

In the following let $\boldsymbol{L}$ be a first-order language containing Peano Arithmetic, a unary relation **nat**, constants for the natural numbers, and the traditional arithmetical operations $+$, $*$, $\leqslant$, $=$ etc. [15]. We use standard notation for composing formulas: $\&$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\exists$ and $\forall$. A (countably infinite) subset of the variables of $\boldsymbol{L}$ will be called *program variables*. An *expression* e is a term in $\boldsymbol{L}$ with only program variables occurring freely whereas a *boolean expression* b is a quantifier free formula of $\boldsymbol{L}$ with program variables as the only free variables.

The language of *while programs* over $\boldsymbol{L}$ is now defined to be the least set of programs satisfying

- for every program variable x and expression e, $x := e$ is a program,
- if c and c' are programs, then so are $c;c'$, IF b THEN c ELSE c' and WHILE b DO c for every boolean expression b.

The free variables of a term of $\boldsymbol{L}$ (a formula of $\boldsymbol{L}$ or a program) will be denoted $FV(\ldots)$.

The semantics of the programs are defined relative to that of $\boldsymbol{L}$. A *semantic model* for $\boldsymbol{L}$ is a structure [15] $\boldsymbol{I}$ satisfying

- the sublanguage of Peano Arithmetic has its standard model [15],
- the symbol **nat** is interpreted as 'is a natural number'.

A *state* s is an assignment of values from the universe of $\boldsymbol{I}$ to the program variables of $\boldsymbol{L}$. The value of an expression e in s is denoted $e(s)$ and the truth of a boolean

expression b in s is written $\models b(s)$. We write $s[v/x]$ for the state that is as s except that x has the value v.

The *semantics* of the programs will be given by a set of axioms and rules defining a relation $\langle c,s\rangle \to s'$ whose intuitive meaning is that the execution of the program c from the state s will terminate and the final state is s'. A similar approach to operational semantics has been used in [5, 14]. The set $\boldsymbol{S}$ of axioms and rules defining the semantics is

$$[\boldsymbol{S}\text{-}1]\quad \langle x := e,s\rangle \to s[e(s)/x]$$

$$[\boldsymbol{S}\text{-}2]\quad \frac{\models b(s),\ \langle c_1,s\rangle \to s'}{\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2,s\rangle \to s'}$$

$$[\boldsymbol{S}\text{-}3]\quad \frac{\models \neg b(s),\ \langle c_2,s\rangle \to s'}{\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2,s\rangle \to s'}$$

$$[\boldsymbol{S}\text{-}4]\quad \frac{\langle c_1,s\rangle \to s',\ \langle c_2,s'\rangle \to s''}{\langle c_1;c_2,s\rangle \to s''}$$

$$[\boldsymbol{S}\text{-}5]\quad \frac{\models b(s),\ \langle c,s\rangle \to s',\ \langle \text{WHILE } b \text{ DO } c,s'\rangle \to s''}{\langle \text{WHILE } b \text{ DO } c,s\rangle \to s''}$$

$$[\boldsymbol{S}\text{-}6]\quad \frac{\models \neg b(s)}{\langle \text{WHILE } b \text{ DO } c,s\rangle \to s}$$

To specify the computation time of the programs we shall extend the semantic model $\boldsymbol{I}$ of $\boldsymbol{L}$ to define the computation time for the expressions and boolean expressions. A *computational model* for $\boldsymbol{L}$ is a semantic model $\boldsymbol{I}$ where

- for each expression e (boolean expression b) there is a total function e^+ (b^+, resp.) that for each state s gives a natural number being the time required to evaluate e in state s (to evaluate b in s, resp.).

This notion of computational model is very simple in that it only takes into account the time required to for evaluating the expression (or boolean expression) and, e.g., ignores the time taken to store a value in memory. More realistic models can easily be defined [4, 13] but we shall refrain from this here as it does not give further insight in relation to the development of a proof system for computation time.

The semantics and computation time of the programs can now be specified by an extension of the relation $\langle c,s\rangle \to s'$ above. The relation $\langle c,s\rangle \xrightarrow{t} s'$ intuitively expresses that the execution of c from s will terminate in s' after exactly t time units. The set $\boldsymbol{S}_t$ of axioms and rules below defines the relation:

$$[\boldsymbol{S}_t\text{-}1]\quad \langle x := e,s\rangle \xrightarrow{e^+(s)} s[e(s)/x]$$

$$[\boldsymbol{S}_t\text{-}2]\quad \frac{\models b(s),\ \langle c_1,s\rangle \xrightarrow{t} s'}{\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2,s\rangle \xrightarrow{b^+(s)+t} s'}$$

$$[\boldsymbol{S}_t\text{-}3]\quad \frac{\models \neg b(s), \langle c_2,s\rangle \xrightarrow{t} s'}{\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2,s\rangle \xrightarrow{b^+(s)+t} s'}$$

$$[\boldsymbol{S}_t\text{-}4]\quad \frac{\langle c_1,s\rangle \xrightarrow{t} s', \langle c_2,s'\rangle \xrightarrow{t'} s''}{\langle c_1;c_2,s\rangle \xrightarrow{t+t'} s''}$$

$$[\boldsymbol{S}_t\text{-}5]\quad \frac{\models b(s), \langle c,s\rangle \xrightarrow{t} s', \langle \text{WHILE } b \text{ DO } c,s'\rangle \xrightarrow{t'} s''}{\langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{b^+(s)+t+t'} s''}$$

$$[\boldsymbol{S}_t\text{-}6]\quad \frac{\models \neg b(s)}{\langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{b^+(s)} s}$$

The obvious relationship between the two relations defined by $\boldsymbol{S}$ and $\boldsymbol{S}_t$ is expressed by

Fact 1. $\langle c,s\rangle \to s'$ if and only if $\langle c,s\rangle \xrightarrow{t} s'$ for some t.

For later reference we shall state a few facts holding for $\boldsymbol{S}_t$ (and thereby $\boldsymbol{S}$). The first expresses that the language is deterministic:

Fact 2. If $\langle c,s\rangle \xrightarrow{t} s'$ and $\langle c,s\rangle \xrightarrow{t'} s''$, then $t = t'$ and $s' = s''$.

The next two facts express that a program c can only modify the values of the variables occurring free in it. For a finite set V of variables define

$$s \equiv_V s' \text{ if and only if } x(s) = x(s') \text{ for all } x \in V.$$

Then

Fact 3. If $\langle c,s\rangle \xrightarrow{t} s'$ and $V \cap FV(c) = \emptyset$, then $s \equiv_V s'$.

Fact 4. If $\langle c,s\rangle \xrightarrow{t} s'$, $FV(c) \subseteq V$ and $s \equiv_V s_0$, then $\langle c,s_0\rangle \xrightarrow{t} s_0'$ for some s_0' with $s_0' \equiv_V s'$.

Note that the computation time of a program is specified by extending the semantic model. This is contrary to the approaches of [11, 16] where the programs are transformed such that they compute the computations time rather than the usual input/output relation.

3. The proof system

The proof system for computation time will be an extension of a proof system for total correctness so we first introduce our formulation of a proof system for total correctness. The basic formulas will have the form $P\langle c\rangle Q/V$ where c is a program, P and Q are formulas of $\boldsymbol{L}$ and V is a (finite) set of program variables. The intuitive meaning of the formula is that if P holds before the execution of c, then c will terminate and Q will hold for the input/output computation performed by c. The component V is included for technical reasons; it can be thought of as the set of program variables we are interested in.

The pre-condition P expresses a property of the initial values of the variables of c and we shall require that P is a *pure formula*; this means that it only has program variables as free variables. The truth of P in the state s is written $\models P(s)$. We have

Fact 5. If $FV(P)\subseteq V$, then $s\equiv_V s_0$ and $\models P(s)$ implies $\models P(s_0)$.

In order to express the post-condition Q we shall distinguish between program variables and some other variables called *shadow variables*. For each program variable x we shall assume that $\boldsymbol{L}$ contains a distinct shadow variable $\tilde{x}$. A relation between the initial and the final values of the variables can now be expressed in a formula of $\boldsymbol{L}$ by letting $\tilde{x}$ refer to the initial value of x, and x itself refer to the final value of x. As an example the formula

$$\exists q.\quad \tilde{x}=q*\tilde{y}+x\ \ \&\ \ x<\tilde{y}$$

expresses that the final value of x is the remainder of the initial value of x divided by the initial value of y (only non-negative numbers are considered). A formula of $\boldsymbol{L}$ with both program and shadow variables as free variables is called a *relational formula*. We shall require that the post-condition Q is a relational formula.

The truth of a relational formula Q is written $\models Q(s,s')$: the state s gives values to the shadow variables whereas s' gives values to the program variables. Thus s and s' can be viewed as the 'shadow part' and the 'program part' of an extended state that gives values to all the variables of L. Define $\tilde{V}=\{\tilde{x}\mid x\in V\}$ for a finite set V of program variables. Corresponding to Fact 5 we have:

Fact 6. If $FV(Q)\subseteq V\cup\tilde{V}$, then $s\equiv_V s_0$, $s'\equiv_V s_0'$ and $\models Q(s,s')$ implies $\models Q(s_0,s_0')$.

A pure formula is also a relational formula and we have

Fact 7. For any pair (s,s'), $\models P(s,s')$ if and only if $\models P(s')$

To simplify the notation used in the axioms and rules of the proof system below we shall introduce some abbreviations. Let X be a vector of the program variables in the (finite) set V and let $\tilde{X}$ be the corresponding vector of shadow variables. For

each pure formula P we define $\tilde{P}$ to be the relational formula $P[\tilde{X}/X]$, that is, the formula obtained by replacing the program variables in V with the corresponding shadow variables. Similarly, for each expression e we write $\tilde{e}$ for $e[\tilde{X}/X]$. We have

Fact 8. If $FV(P) \subseteq V$, then for any pair (s,s'), $\models P(s)$ if and only if $\models \tilde{P}(s,s')$; furthermore, $e(s) = x(s')$ if and only if $\models x = \tilde{e}(s,s')$.

For two relational formulas Q_1 and Q_2 we write $Q_1 \cdot Q_2$ as an abbreviation for the formula

$$\exists X'. \quad Q_1[X'/X] \ \& \ Q_2[X'/\tilde{X}]$$

where X' is a vector of distinct new variables of the same length as X (and $\tilde{X}$). If $FV(Q_i) \subseteq V \cup \tilde{V}$ for $i = 1, 2$, then $Q_1 \cdot Q_2$ expresses the composition of the relations specified by Q_1 and Q_2 [1]. This is captured by

Fact 9. If $FV(Q_i) \subseteq V \cup \tilde{V}$ for $i = 1, 2$, then for any pair (s,s'), $\models Q_1 \cdot Q_2(s,s')$ if and only if $\models Q_1(s,s'')$ and $\models Q_2(s'',s')$ for some s''.

Finally, we write I_V as an abbreviation of the relational formula $\ldots \& x = \tilde{x} \& \ldots$ (for every x in V). Thus

Fact 10. $\models I_V(s,s')$ if and only if $s \equiv_V s'$.

Recall that the *formulas* of the proof system are constructs of the form $P\langle c\rangle Q/V$ where c is a program, P a pure formula, Q a relational formula and V a finite set of program variables. We shall impose a *well-formedness condition* on the formulas reflecting the intuition that V is the set of variables we are interested in:

$$FV(c) \subseteq V, \qquad FV(P) \subseteq V \quad \text{and} \quad FV(Q) \subseteq V \cup \tilde{V}.$$

The *validity* of a well-formed formula $P\langle c\rangle Q/V$ (in the semantic model $\boldsymbol{I}$) is defined by

for any state s, if $\models P(s)$, then $\langle c,s\rangle \to s'$ and $\models Q(s,s')$ for some s'.

We shall write $\models_{\boldsymbol{I}} P\langle c\rangle Q/V$ or, omitting the subscript $\boldsymbol{I}$, $\models P\langle c\rangle Q/V$.

The axioms and rules of the *proof system* $\boldsymbol{T}$ for total correctness are:

[$\boldsymbol{T}$-1] $P\langle x := e\rangle I_{V-\{x\}} \& x = \tilde{e}/V$

[$\boldsymbol{T}$-2] $$\frac{P\&b\langle c_1\rangle Q/V,\ P\&\neg b\langle c_2\rangle Q/V}{P\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2\rangle Q/V}$$

[$\boldsymbol{T}$-3] $$\frac{P\langle c_1\rangle P'\&Q_1/V,\ P'\langle c_2\rangle Q_2/V}{P\langle c_1;c_2\rangle Q_1 \cdot Q_2/V}$$

[$\boldsymbol{T}$-4] $$\frac{\begin{array}{c} P(z+1)\&b\langle c\rangle P(z)\&Q'/V \cup \{z\} \\ P(0) \to \neg b,\ P(z)\&\neg b\&I_V \to Q,\ Q' \cdot Q \to Q \end{array}}{\exists z.\, P(z)\langle \text{WHILE } b \text{ DO } c\rangle Q/V}$$

where z (not in V) is a program variable ranging over the natural numbers

$$[\boldsymbol{T}\text{-}5]\ \frac{P \to P',\ P'\langle c\rangle Q'/V,\ Q' \to Q}{P\langle c\rangle Q/V}$$

$$[\boldsymbol{T}\text{-}6]\ \frac{P\langle c\rangle Q/V}{P\langle c\rangle \tilde{P}\&Q/V}.$$

We shall write $\boldsymbol{T} \vdash_{\boldsymbol{I}} P\langle c\rangle Q/V$ (or, omitting the subscript $\boldsymbol{I}$, $\boldsymbol{T} \vdash P\langle c\rangle Q/V$) if the formula $P\langle c\rangle Q/V$ is *provable* using the axioms and rules above together with the formulas of $\boldsymbol{L}$ being valid in the semantic model $\boldsymbol{I}$.

Except for the while rule this proof system is essentially as that suggested by Manna and Pnueli [10]. Consequently, we refer to [10] (or the discussion following the proof system $\boldsymbol{R}$ below) for an explanation of the axioms and rules. In the while rule termination is ensured as suggested by Harel [7]. Some of Aczel's notational abbreviations [1] have been adopted in order to simplify (the appearance of) the proof system. The component V has been introduced in the formulas $P\langle c\rangle Q/V$ in order to express the post-conditions sufficiently precisely, e.g. in the axiom for the assignment where all variables except one is unchanged. To keep within first-order logic we shall only allow a finite set V of 'interesting' program variables. We have chosen to supply V as a part of the formulas instead of parameterizing the whole proof system on V. The actual choice is not of profound importance although it affects the soundness and completeness proofs. Note that the consideration of V is not relevant for [1, 8] because post-conditions are relations and not formulas, and is not considered in [10] because completeness is not studied.

We now extend the proof system to prove properties of the computation time. A property of the computation time of a program is a relation between the (initial) values of its variables and the corresponding computation time. We shall therefore define a *time formula* to be a formula of $\boldsymbol{L}$ that has the special variable **time** (ranging over the natural numbers) and program variables as permitted free variables. The variable **time** will neither be a program variable nor a shadow variable; intuitively, it denotes the (unknown) computation time. The truth of a time formula R in state s with **time** being t is written $\models R(s,t)$. We have

Fact 11. If $FV(R) \subseteq V \cup \{\textbf{time}\}$, then $s =_V s'$ and $\models R(s,t)$ implies $\models R(s',t)$.

A pure formula P is also a time formula and we have

Fact 12. For any s, $\models P(s)$ if and only if for every t, $\models P(s,t)$.

To simplify the notation used in the axioms and rules below we shall introduce a few shorthands. The composition of a relational formula Q and a time formula R, written $Q \cdot R$, is defined by

$$\exists X'.\quad Q[X'/X][X/\tilde{X}]\ \&\ R[X'/X]$$

(X' is a vector of distinct new variables of the same length as X). If $FV(Q) \subseteq V \cup \tilde{V}$ and $FV(R) \subseteq V \cup \{\mathbf{time}\}$, then $Q \cdot R$ is the composition of the relations specified by Q and R:

Fact 13. If $FV(Q) \subseteq V \cup \tilde{V}$ and $FV(R) \subseteq V \cup \{\mathbf{time}\}$, then for any pair (s,t), $\models Q \cdot R(s,t)$ if and only if $\models Q(s,s')$ and $\models R(s',t)$ for some s'.

Two time formulas R_1 and R_2 can be 'added' as follows: $R_1 \oplus R_2$ is an abbreviation for the formula

$$\exists t' \exists t''. \quad \mathbf{time} = t' + t'' \ \&\ R_1[t'/\mathbf{time}] \ \&\ R_2[t''/\mathbf{time}].$$

We have

Fact 14. For any pair (s,t), $\models R_1 \oplus R_2(s,t)$ if and only if $\models R_1(s,t')$ and $\models R_2(s,t'')$ for some t' and t'' with $t = t' + t''$.

In order to *formulate* the proof system $\boldsymbol{R}$ for analysing computation time, we shall impose an expressiveness condition on $\boldsymbol{L}$ and the computational model $\boldsymbol{I}$ ensuring that we have terms for the time requirements of the expressions and boolean expressions of the programs. The *time expressiveness condition* is fulfilled if

- for every expression e there is a term $T[e]$ with $FV(T[e]) \subseteq FV(e)$ such that for every state s, $T[e](s) = e^+(s)$, and
- for every boolean expression b there is a term $T[b]$ with $FV(T[b]) \subseteq FV(b)$ such that for every state s, $T[b](s) = b^+(s)$.

From this definition it follows that

Fact 15. If $FV(e) \subseteq V$ and $s \equiv_V s'$, then $T[e](s) = T[e](s')$ and if $FV(b) \subseteq V$ and $s \equiv_V s'$, then $T[b](s) = T[b](s')$.

Note that the time expressiveness condition is a property of the first-order language $\boldsymbol{L}$ and its computational model $\boldsymbol{I}$, and as such it is independent of the programming language defined on top of $\boldsymbol{L}$. This is contrary to Cook's notion of expressiveness [3, 6] used when proving the completeness of proof systems for partial correctness.

The *formulas* of the proof system $\boldsymbol{R}$ has the form $P\langle c : R\rangle Q / V$ where P, c, Q and V are as in $\boldsymbol{T}$ and R is a time formula. The *well-formedness condition* is now defined to mean that

$$FV(c) \subseteq V, \qquad FV(P) \subseteq V, \qquad FV(Q) \subseteq V \cup \tilde{V} \text{ and } FV(R) \subseteq V \cup \{\mathbf{time}\}.$$

Given a computational model $\boldsymbol{I}$ for $\boldsymbol{L}$ the *validity* of a well-formed formula $P\langle c : R\rangle Q / V$, written $\models P\langle c : R\rangle Q / V$ (omitting the subscript $\boldsymbol{I}$), is defined to mean that

for every state s, if $\models P(s)$, then

$\langle c,s\rangle \xrightarrow{t} s'$, $\models Q(s,s')$ and $\models R(s,t)$ for some s' and t.

Obviously, we have the following relationship between the formulas of $\boldsymbol{T}$ and $\boldsymbol{R}$:

Fact 16. $\models P\langle c: R\rangle Q/V$ implies $\models P\langle c\rangle Q/V$
(using Fact 1 and that a computational model also is a semantic model).

The axioms and rules of $\boldsymbol{R}$ are proper extensions of those of $\boldsymbol{T}$:

$$[\boldsymbol{R}\text{-}1]\quad P\langle x := e : \mathbf{time} = T[e]\rangle I_{V-\{x\}} \& x = \tilde{e}/V$$

$$[\boldsymbol{R}\text{-}2]\quad \frac{P\&b\langle c_1: R\rangle Q/V,\ P\&\neg b\langle c_2: R\rangle Q/V}{P\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 : (\mathbf{time} = T[b])\oplus R\rangle Q/V}$$

$$[\boldsymbol{R}\text{-}3]\quad \frac{P\langle c_1: R_1\rangle P' \& Q_1/V,\ P'\langle c_2: R_2\rangle Q_2/V}{P\langle c_1; c_2 : R_1 \oplus (Q_1 \cdot R_2)\rangle Q_1 \cdot Q_2/V}$$

$$[\boldsymbol{R}\text{-}4]\quad \frac{\begin{array}{c} P(z+1)\&b\langle c: R'\rangle P(z)\,\&\, Q'/V\cup\{z\} \\ P(0)\to\neg b,\ P(z)\&\neg b\,\&\, I_V \to Q,\ Q'\cdot Q\to Q \\ P(z)\&\neg b\,\&(\mathbf{time} = T[b])\to R,\ (\mathbf{time} = T[b])\oplus R'\oplus(Q'\cdot R)\to R \end{array}}{\exists z.\, P(z)\langle \text{WHILE } b \text{ DO } c: R\rangle Q/V}$$

where z (not in V) is a program variable ranging over the natural numbers

$$[\boldsymbol{R}\text{-}5]\quad \frac{P\to P',\ P'\langle c: R'\rangle Q'/V,\ Q'\to Q,\ R'\to R}{P\langle c: R\rangle Q/V}$$

$$[\boldsymbol{R}\text{-}6]\quad \frac{P\langle c: R\rangle Q/V}{P\langle c: P\&R\rangle \tilde{P}\&Q/V}.$$

We shall write $\boldsymbol{R}\vdash P\langle c:R\rangle Q/V$ (omitting the subscript $\boldsymbol{I}$) if the formula $P\langle c:R\rangle Q/V$ is *provable* using the axioms and rules above together with the formulas of $\boldsymbol{L}$ being valid in the computational model $\boldsymbol{I}$. It is easy to show that

Fact 17. $\boldsymbol{R}\vdash P\langle c:R\rangle Q/V$ implies $\boldsymbol{T}\vdash P\langle c\rangle Q/V$.

In the axiom [$\boldsymbol{R}$-1] the post-condition $I_{V-\{x\}}\&x=\tilde{e}$ expresses that all the variables of V except x are unchanged by the assignment and that the value of x now is equal to the 'old' value of e. The time formula $\mathbf{time} = T[e]$ reflects that the time taken to execute $x := e$ is given by $T[e]$.

In [$\boldsymbol{R}$-2] we simply 'add' the time taken to evaluate the condition to the time formula for the branches.

In the rule [$\boldsymbol{R}$-3] for composition we assume that the post-condition of c_1 has been split into a pure part P' and a relational part Q_1. The composition of Q_1 with the post-condition Q_2 of c_2 describes the effect of executing $c_1;c_2$. The formula R_2 expresses a relation between the values of the variables before c_2 is executed and

the computation time for c_2. Since Q_1 describes the effect of executing c_1 we get that $Q_1 \cdot R_2$ expresses a relation between the initial values of the variables (before c_1 is executed) and the computation time of c_2. The time formula R_1 is assumed to be a property of the computation time of c_1 so $R_1 \oplus (Q_1 \cdot R_2)$ will be a property of the computation time of $c_1;c_2$.

In the rule [***R***-4] for the while construct the idea is that the new variable z bounds the number of unfoldings of the loop. The post-condition of the body is split into a pure part (the invariant) expressing that now fewer unfoldings are required and a relational part Q' describing the effect of executing the body once. The relational formula Q describes the effect of executing the body a number of times—this is ensured by the two assumptions $P(z) \& \neg b \& I_V \rightarrow Q$ and $Q' \cdot Q \rightarrow Q$. The time formula R' is assumed to be a property of the computation time for one execution of the body of the loop whereas R is a property of the computation time of a number of unfoldings of the loop. The two assumptions $P(z) \& \neg b \& (\mathbf{time} = T[b]) \rightarrow R$ and $(\mathbf{time} = T[b]) \oplus R' \oplus (Q' \cdot R) \rightarrow R$ ensure that this is indeed the case because Q' describes the effect of executing the body once. Intuitively, the two assumptions express that R is a solution to a certain *recurrence relation* [2]; this will be illustrated by an example in Section 5.

The rule of consequence [***R***-5] is a straightforward extension of the traditional rule [3] whereas the rule of invariance [***R***-6] mainly is due to the use of relations as post-conditions.

4. Soundness and completeness results

We now formulate and prove soundness and completeness results for the systems ***T*** and ***R***. We concentrate on ***R*** as the results for ***T*** may be obtained as corollaries for those of ***R***. The actual details in the proof may be found in Appendix A and in this section we only explain the overall structure of the proof.

Following [7] we define an *arithmetical semantic model* to be a semantic model ***I*** with

> a formula ϕ in ***L*** which, when interpreted in ***I***, allows finite sequences of elements from ***I*** to be encoded as a single element of ***I***.

More precisely, ϕ is assumed to be a pure formula with three free variables x, i and y. If $v_1, \ldots, v_k$ is a finite sequence of elements, then they are encoded as the single element u provided that for every natural number n,

$$\models \phi(s[u/y][n/i]) \text{ if and only if } x(s) = v_n.$$

An *arithmetical computational model* is a computational model with an underlying arithmetical semantic model. The soundness and completeness result for ***R*** can now be formulated as

Theorem 4.1. *Given a first-order language* $\mathbf{L}$ *and an arithmetical computational model* $\mathbf{I}$ *such that the time expressiveness condition is fulfilled. Then for every well-formed formula* $P\langle c:R\rangle Q/V$,

$$\mathbf{R} \vdash P\langle c:R\rangle Q/V \text{ if and only of } \models P\langle c:R\rangle Q/V.$$

The *soundness* result is shown by induction in the length of the proofs in $\mathbf{R}$. It is sufficient to show that the axioms of $\mathbf{R}$ are valid and that the rules preserve validity. In most cases this is fairly straightforward using the facts listed earlier and that the time expressiveness condition is fulfilled. In the case of the while rule [$\mathbf{R}$-4] we use that its conclusion is well-formed so that the variable z neither occurs free in the program WHILE b DO c nor in the post-condition Q nor in the time formula R. This means that the value of z in the state is not important for the termination of the loop nor for the truth of the post-condition nor for the truth of the time formula (formally expressed by the Facts 4, 6 and 11) and this is central for the soundness proof.

In the *completeness* proof we may make use of the following lemma stating a sort of expressiveness property for $\mathbf{L}$ and $\mathbf{I}$:

Lemma 4.2. *For every program* c *and finite set* V *of program variables with* $FV(c) \subseteq V$ *there exists three formulas* $P_V(c)$, $G_V(c)$ *and* $E_V(c)$ *satisfying*

(i) $P_V(c)$ *is a pure formula with* $FV(P_V(c)) \subseteq V$ *such that for any state* s

$$\models P_V(c)(s) \text{ if and only if } \langle c,s\rangle \xrightarrow{t} s' \text{ for some } t \text{ and } s';$$

(ii) $G_V(c)$ *is a relational formula with* $FV(G_V(c)) \subseteq V \cup \tilde{V}$ *such that for any pair* (s,s') *of states*

$$\models G_V(c)(s,s') \text{ if and only if } \langle c,s\rangle \xrightarrow{t} s'' \text{ for some } t \text{ and } s'' \text{ with } s'' \equiv_V s'.$$

(iii) $E_V(c)$ *is a time formula with* $FV(E_V(c)) \subseteq V \cup \{\mathbf{time}\}$ *such that for any pair* (s,t) *of state and natural number*

$$\models E_V(c)(s,t) \text{ if and only if } \langle c,s\rangle \xrightarrow{t} s' \text{ for some } s'.$$

The proof of this result can be found in Appendix A. It is by structural induction on the programs. The cases of assignment, conditional and composition are fairly straightforward. In the case of iteration the formula ϕ (whose existence is ensured because we only consider arithmetical computational models) is used to encode the V-part of a finite sequence of states as a single vector and also to encode a finite sequence of computation times as a single number.

Similar formulas are shown to exist in [7] within the framework of first-order dynamic logic. The existence of $G_V(c)$ implies the existence of strongest post-conditions. The strongest post-condition $SP(P,c)$ for a (pure) formula P can be defined by $\exists \tilde{X}.\tilde{P}\&G_V(c)$. Note that $P_V(c)$ expresses a (sort of) weakest precondition.

The completeness proof also make use of the following lemma proved in Appendix A:

Lemma 4.3. *For every well-formed formula* $P\langle c\rangle Q/V$, *if* $\models P\langle c\rangle Q/V$, *then*

$$\models P \to P_V(c), \qquad \models \tilde{P}\&G_V(c) \to Q \quad \textit{and} \quad \models P\&E_V(c) \to R.$$

It is now easy to verify that if $\models P\langle c{:}R\rangle Q/V$ and if we have a proof of $P_V(c)\langle c{:}E_V(c)\rangle G_V(c)/V$ in $\boldsymbol{R}$, then $\boldsymbol{R} \vdash P\langle c{:}R\rangle Q/V$: first we apply [$\boldsymbol{R}$-5] with $\models P \to P_V(c)$, then [$\boldsymbol{R}$-6] and finally [$\boldsymbol{R}$-5] with $\models \tilde{P}\&G_V(c) \to Q$ and $\models P\&E_V(c) \to R$. Thus the completeness result follows from

Lemma 4.4. *For every program* c *and finite set* V *of program variables with* $FV(c) \subseteq V$,

$$\boldsymbol{R} \vdash P_V(c)\langle c{:}E_V(c)\rangle G_V(c)/V.$$

This result is shown in Appendix A by structural induction on the program c. The cases of assignment, conditional and composition are fairly straightforward. In the case of the while loop WHILE b DO c we shall transform the program in order to define the invariant $P'(z)$ bounding the number of unfoldings of the loop. So let z be a program variable not in V and define

$$c' \;=\; \text{IF } z=0 \text{ THEN } \textit{loop} \text{ ELSE } (c;\, z := z-1)$$

where *loop* is a program that always loops and $z := z-1$ decrements the value of z by one. Then $P'(z)$ can be defined to express the termination of the modified program: $P_{V\cup\{z\}}(\text{WHILE } b \text{ DO } c')$. The semantic relationship between the transformed program and the original one is used extensively in the proof.

In the proof of the completeness result given here there is no need for additional variables (except z) in the proof for the while loop. This is contrary to the completeness proofs for the usual proof systems for total correctness where in certain cases new variables are introduced to take snapshots of the program variables [3]. But we always have access to the initial values of the variables thanks to the presence of the shadow variables.

The soundness and completeness result for $\boldsymbol{T}$ can be obtained as a corollary to that for $\boldsymbol{R}$:

Corollary 4.5. *Given a first-order language* $\boldsymbol{L}$ *and an arithmetical semantic model* $\boldsymbol{I}$ *for it, then for every well-formed formula* $P\langle c\rangle Q/V$

$$\boldsymbol{T} \vdash P\langle c\rangle Q/V \textit{ if and only if } \models P\langle c\rangle Q/V.$$

To see this we first extend an arbitrary semantic model $\boldsymbol{I}$ for $\boldsymbol{L}$ to a computational model $\boldsymbol{I}'$ by defining $e^+(s) = 0$ for each expression e and $b^+(s) = 0$ for each boolean expression b. Then Fact 1 can be replaced by

$$\langle c,s\rangle \to s' \text{ if and only if } \langle c,s\rangle \xrightarrow{0} s'.$$

Since 0 is a term of $\boldsymbol{L}$, $\boldsymbol{L}$ and $\boldsymbol{I}'$ fulfill the time expressiveness condition. It is easy to verify that Facts 16 and 17 can be strengthened to

$$\models P\langle c\text{:}\mathbf{time} = 0\rangle Q/V \text{ if and only if } \models P\langle c\rangle Q/V$$

and

$$\boldsymbol{R} \vdash P\langle c\text{:}\mathbf{time} = 0\rangle Q/V \text{ if and only if } \boldsymbol{T} \vdash P\langle c\rangle Q/V$$

resp. The corollary now follows from these two results and the soundness and completeness theorem for $\boldsymbol{R}$.

5. Applications

In the Introduction we mentioned that a proof system for computation time should allow natural formalizations of the traditional informal analyses of computation time. The aim of this section is to discuss to what extent this is the case for the proof system $\boldsymbol{R}$. We shall begin by considering an example, the bubble sorting algorithm.

Consider the following algorithm for sorting an array A of elements:

$i := \mathit{length}(A)$;
WHILE $\neg i = 0$ DO ($m := i$; $i := 0$; $j := 1$;
WHILE $\neg j = m$ DO (IF $A[j] > A[j+1]$
THEN ($A := \mathit{swop}(A, j, j+1)$;
$i := j$; $j := j+1$)
ELSE $j := j+1$)).

We shall not specify the first-order language $\boldsymbol{L}$ and its (arithmetical) computational model in detail but merely mention that the array A and the (non-negative) integers m, i and j are data and that the function *swop* takes an array A and two distinct indices j and k as arguments and returns an array that is as A except that the elements corresponding to the jth and the kth indices have been exchanged. For the sake of simplicity we shall employ a computation model $\boldsymbol{I}$ where *swop* takes one time unit and the remaining operations take no time. Clearly, the time expressiveness condition will be fulfilled for this choice of $\boldsymbol{L}$ and $\boldsymbol{I}$.

An *informal analysis* of the computation time of this algorithm proceeds as follows [2]: First consider the conditional. Each of the branches takes constant time and so does the test. The whole construct therefore takes constant time, that is, it takes time $\mathcal{O}(1)$. The body of the inner loop is executed $m-1$ times, each time the computation time for both the test and the body is $\mathcal{O}(1)$, so the total time for the inner loop is

$$(m-1) * \mathcal{O}(1) = \mathcal{O}(m).$$

The body of the outer loop is executed at most n ($= length(A)$) times and the value of the variable i is decremented each time. The computation time for the body is $\mathcal{O}(i)$ and for the test it is constant. The total time for the outer loop and thereby the complete program is

$$\sum_{i=1}^{n} \mathcal{O}(i) = \mathcal{O}(n^2).$$

Let us now construct a *formal proof* of this result in ***R***. More precisely, we shall prove the formula

$$\text{TRUE } \langle \text{bubble-sorting: } \mathbf{time} \leqslant length(A)^2 \rangle \text{ TRUE}/V \qquad (*)$$

where $V = \{A, i, j, m\}$. The pre- and post-conditions of the formula are chosen to be the formula TRUE as we are mainly interested in the computation time of the program. ((Thus we do not prove that the algorithm sorts the array.)

The proof will be presented in a bottom-up manner so we start with the conditional. Using the rules [***R***-1], [***R***-3], [***R***-5] and [***R***-6] we obtain proofs of the two formulas

$$P'(z'+1)\&\neg j = m\&A[j] > A[j+1] \ \langle \text{true-branch: } \mathbf{time} \leqslant 1 \rangle \ P'(z')\&m = \tilde{m}\&z = \tilde{z}\&j = \tilde{j}+1/V \cup \{z,z'\}$$

and

$$P'(z'+1)\&\neg j = m\&\neg A[j] > A[j+1] \ \langle \text{false-branch: } \mathbf{time} \leqslant 1 \rangle \ P'(z')\&m = \tilde{m}\&z = \tilde{z}\&j = \tilde{j}+1/V \cup \{z,z'\}$$

where

$$P'(z') = (0 < m < z+1 \,\&\, z'+j = m \,\&\, i < j)$$

is the invariant for the inner loop and z is the counter introduced for the outer loop. Using the rule [***R***-2] we get a proof of

$$P'(z'+1)\&\neg j = m \ \langle \text{body-of-inner-loop: } \mathbf{time} \leqslant 1 \rangle \ P'(z')\&m = \tilde{m}\&z = \tilde{z}\&j = \tilde{j}+1/V \cup \{z,z'\}$$

since $T[A[j] > A[j+1]]$ is the constant 0.

The next step is to apply the rule [***R***-4] in order to get a proof for the inner loop. We have

$$\models P'(0) \rightarrow j = m,$$
$$\models P'(z')\&j = m\&I_{V\cup\{z\}} \rightarrow i < m\&m = \tilde{m}\&z = \tilde{z}$$

and

$$\models (m = \tilde{m}\&z = \tilde{z}\&j = \tilde{j}+1)\cdot(i < m\&m = \tilde{m}\&z = \tilde{z}) \rightarrow (i < m\&m = \tilde{m}\&z = \tilde{z}).$$

The invariant for the computation time of the loop is specified by the time formula $\mathbf{time} \leqslant m - j$ and using that $T[\neg j = m]$ is 0 we have

$$\models P'(z')\&j = m\&\mathbf{time} = 0 \rightarrow \mathbf{time} \leqslant m - j$$

and

$$\models (\mathbf{time}=0)\oplus(\mathbf{time}\leqslant 1)$$
$$\oplus((m=\tilde{m}\&z=\tilde{z}\&j=\tilde{j}+1)\cdot(\mathbf{time}\leqslant m-j))\ \rightarrow\ (\mathbf{time}\leqslant m-j)$$

Applying [***R***-4] we now get a proof of

$$\exists z'.P'(z')\ \langle\text{inner-loop: } \mathbf{time}\leqslant m-j\rangle\ i<m\&m=\tilde{m}\&z=\tilde{z}/\,V\cup\{z\}.$$

Using the rules [***R***-1], [***R***-3], [***R***-5] and [***R***-6] we can now obtain a proof of the formula

$$P(z+1)\&\neg i=0\ \langle\text{body-of-outer-loop: } \mathbf{time}\leqslant i-1\rangle\ P(z)\&i<\tilde{i}/\,V\cup\{z\}$$

where

$$P(z)=i\leqslant z$$

is the invariant for the loop. The next step is to apply the rule [***R***-4] to the outer loop. We have

$$\models P(0)\ \rightarrow\ i=0,$$
$$\models P(z)\&i=0\&I_V\ \rightarrow\ \text{TRUE}$$

and

$$\models (i<\tilde{i})\cdot\text{TRUE}\ \rightarrow\ \text{TRUE}.$$

For the invariant for the computation time of the loop we use the formula $\mathbf{time}\leqslant i^2$. We then have

$$\models P(z)\&i=0\&\mathbf{time}=0\ \rightarrow\ \mathbf{time}\leqslant i^2$$

and

$$\models(\mathbf{time}=0)\oplus(\mathbf{time}\leqslant i-1)\oplus((i<\tilde{i})\cdot(\mathbf{time}\leqslant i^2))\ \rightarrow\ (\mathbf{time}\leqslant i^2).$$

Since $T[\neg i=0]$ is 0 we get a proof of

$$\exists z.P(z)\ \langle\text{outer-loop: } \mathbf{time}\leqslant i^2\rangle\ \text{TRUE}/\,V.$$

It is now straightforward to obtain the required proof of (*).

Comparing the informal analysis to the formal proof in ***R*** we note that to a large extent they *proceed in the same way.* In both cases we have associated a computation time property with each piece of program and composed them while working our way inside out to get a property of the computation time for the complete program.

However, the computation time properties are composed in somewhat different ways. Consider for instance the outer loop. The informal analysis follows the informal rule stated earlier: "the computation time of a loop is the sum, over all the times round the loop, of the time to execute the body and the time for evaluating

the condition". Thus we calculate the sum

$$\sum_{i=1}^{n} \mathcal{O}(i)$$

in the informal proof. The situation is slightly different in the formal proof. We have a proof showing that the formula $\mathbf{time} \leqslant i-1$ holds for the computation time of the body of the outer loop. In order to apply the rule [***R***-4] we have to find a time formula R such that

$$\begin{aligned} &\models P(z) \& i = 0 \& \mathbf{time} = 0 \rightarrow R, \quad \text{and} \\ &\models (\mathbf{time} = 0) \oplus (\mathbf{time} \leqslant i-1) \oplus ((i < \tilde{i}) \cdot R) \rightarrow R. \end{aligned} \tag{**}$$

If we restrict the formula R to have the form $\mathbf{time} \leqslant T(i)$ where $T(i)$ is a term with the only free variable i, then we can replace (**) by

$$T(0) = 0,$$

$$T(\tilde{i}) + (i-1) \leqslant T(i) \quad \text{where } \tilde{i} < i$$

and the problem will be to find a (minimal) solution to this recurrence relation. Further elaboration shows that it is sufficient to find a (minimal) solution to the recurrence relation

$$T(0) = 0,$$

$$T(i) = (i-1) + T(i-1)$$

and this is exactly the summing series

$$T(n) = \sum_{i=0}^{n} i - 1 \sim \sum_{i=1}^{n} \mathcal{O}(i)$$

of the informal analysis. Thus by specializing the form of the time formulas we obtain a recurrence equation whose solution is the summing series of the informal analysis. We shall conclude that the informal analysis has been formalized in a reasonably direct way in ***R***.

To substantiate this claim further we shall show how formalized versions of the informal rules can be derived in ***R***. The informal rules are mainly used for proving *upper bounds* on the computation time. We shall therefore restrict the time formulas R of the formulas $P\langle c:R\rangle Q / V$ to have the form $\mathbf{time} \leqslant T$ where T is a term of ***L*** with program variables as the only permitted free variables.

In order to do this we shall first assume that the computation times for the expressions and boolean expressions are bounded by a constant since this is an implicit assumption in the informal rules. That is we assume:

- for every expression e there exists a natural number $K[e]$ such that for every state s we have $e^{+}(s) \leqslant K[e]$, and
- for every boolean expression b there exists a natural number $K[b]$ such that for every state s we have $b^{+}(s) \leqslant K[b]$.

First consider the *assignment* statement where the rule says that the time is bounded by a constant. Obviously, $\models \mathbf{time} = T[e] \rightarrow \mathbf{time} \leq K[e]$ so using [***R***-1] and [***R***-5] we get a proof of

$$[\text{ass}] \qquad P\langle x := e{:}\mathbf{time} \leq K[e]\rangle I_{V-\{x\}} \& x = \tilde{e}/V$$

which formalizes the informal rule.

The informal rule for *composition* says that the time, to within a constant factor, is the largest of the computation times for the two statements. So assume that we have proofs of $P\langle c_1{:}\mathbf{time} \leq T_1\rangle P' \& Q_1/V$ and $P'\langle c_2{:}\mathbf{time} \leq T_2\rangle Q_2/V$. Using [***R***-3] we then get a proof of $P\langle c_1; c_2{:}(\mathbf{time} \leq T_1) \oplus (Q_1 \cdot (\mathbf{time} \leq T_2))\rangle Q_1 \cdot Q_2/V$. We shall now replace the formula $(\mathbf{time} \leq T_1) \oplus (Q_1 \cdot (\mathbf{time} \leq T_2))$ by $\mathbf{time} \leq T$ where T is the 'sum' of T_1 and T_2. This can e.g. be achieved by requiring that $Q_1 \rightarrow \tilde{T}_1 + \tilde{T}_2 \leq \tilde{T}$ since then $(\mathbf{time} \leq T_1) \oplus (Q_1 \cdot \mathbf{time} \leq T_2) \rightarrow \mathbf{time} \leq T$. So we obtain the rule

$$[\text{comp}] \qquad \frac{P\langle c_1{:}\mathbf{time} \leq T_1\rangle P' \& Q_1/V,\ P'\langle c_2{:}\mathbf{time} \leq T_2\rangle Q_2/V \quad Q_1 \rightarrow \tilde{T}_1 + T_2 \leq \tilde{T}}{P\langle c_1; c_2{:}\mathbf{time} \leq T\rangle Q_1 \cdot Q_2/V}$$

This is not exactly a formalization of the informal rule because we use the 'sum' of T_1 and T_2 rather than the 'largest' of them. The reason for this is that the informal rule ignores constant factors.

The first informal rule for *iteration* states that the time for the loop is the sum, over all times round the loop, of the time to execute the body and the time for evaluating the condition. This rule may be formalized by

$$[\text{itr1}] \qquad \frac{\begin{array}{c} P(z+1) \& b\langle c{:}\mathbf{time} \leq T'\rangle P(z) \& Q'/V \cup \{z\} \\ P(0) \rightarrow \neg b,\ P(z) \& \neg b \& I_V \rightarrow Q,\ Q' \cdot Q \rightarrow Q \\ P(z) \& \neg b \rightarrow K[b] \leq T,\ Q' \rightarrow \widetilde{K[b]} + \tilde{T}' + T \leq \tilde{T} \end{array}}{\exists z.P(z)\langle \text{WHILE } b \text{ DO } c{:}\mathbf{time} \leq T\rangle Q/V}$$

where z (not in V) is a program variable ranging over the natural numbers

Note that this rule does not directly give the summing series we might expect from the informal rule but rather the recurrence equations that define it. To see that [itr1] can be derived in ***R*** we first observe that the assumptions about the computational model give us that $\models P(z) \& \neg b \& \mathbf{time} = T[b] \rightarrow \mathbf{time} \leq T$. The assumption $Q' \rightarrow \widetilde{K[b]} + \tilde{T}' + T \leq \tilde{T}$ can be used to show that $\models (\mathbf{time} = T[b]) \oplus (\mathbf{time} \leq T') \oplus (Q' \cdot (\mathbf{time} \leq T)) \rightarrow \mathbf{time} \leq T$. The rule [***R***-4] then gives the conclusion of [itr1] directly.

The second informal rule for iteration states that the computation time of the loop also can be bounded by the number of times the loop is executed and the

maximal computation time for the body of the loop. In the following let B be a term in L intended to bound the number of times the loop is executed. A possible rule then is

$$[\text{itr2}]\quad \frac{\begin{array}{c} P(z+1)\&b\langle c{:}\mathbf{time} \leqslant T\rangle P(z)\&Q'/V\cup\{z\} \\ P(0)\to\neg b,\ P(z)\&\neg b\&I_V\to Q,\ Q'\cdot Q\to Q \\ P(z)\&\neg b\to 0\leqslant B,\ Q'\to T\leqslant\tilde{T}\&B<\tilde{B} \end{array}}{\exists z.P(z)\langle\text{WHILE } b \text{ DO } c{:}\mathbf{time}\leqslant B*(K[b]+T)+K[b]\rangle Q/V}$$

where z (not in V) is a program variable ranging over the natural numbers

The assumption $Q'\to T\leqslant\tilde{T}\&B<\tilde{B}$ ensures that B is decremented for each time the loop is executed and that the term T can be used to bound the execution time for each of the executions of the body. To derive the rule in $\boldsymbol{R}$ first observe that $\models P(z)\&\neg b\&\mathbf{time}=T[b]\to\mathbf{time}\leqslant B*(K[b]+T)+K[b]$ because the computation time for b is bounded by $K[b]$ and we have assumed that $P(z)\&\neg b\to 0\leqslant B$. From $Q'\to T\leqslant\tilde{T}\&B<\tilde{B}$ we get $\models(\mathbf{time}=T[b])\oplus(\mathbf{time}\leqslant T)\oplus(Q'\cdot(\mathbf{time}\leqslant B*(K[b]+T)+K[b]))\to\mathbf{time}\leqslant B*(K[b]+T)+K[b]$ so by applying [$\boldsymbol{R}$-4] we get a proof of the conclusion of [itr2].

To summarize we claim that a large number of the informal analyses of, e.g., [2] can be formalized as proofs in $\boldsymbol{R}$ in this rather direct way. Of course there are limitations in that not all analyses proceed in a structural way. An example is Dijkstra's graph algorithm for solving the single source shortest path problem [2]. Here the informal rules do not suffice to prove the $\mathcal{O}(e+n*\log n)$ bound on the computation time because a special bookkeeping trick is used. Essentially, the main loop of the algorithm is analysed twice: once to calculate the computation time charged to the number n of nodes in the graph and once to calculate that charged to the number e of edges. The completeness result for $\boldsymbol{R}$ shows that the $\mathcal{O}(e+n*\log n)$ upper bound can indeed be proved in $\boldsymbol{R}$ but we cannot expect the actual proof to be close to the informal one because the former has to proceed in a structural way and the latter does not. However, if the informal analyses are based on the informal rules [2] mentioned in the Introduction, then we claim that they can be formalized rather directly.

6. Conclusion

We have shown how a proof system for total correctness can be extended to prove properties of the computation time. The notions of arithmetical soundness and completeness have been extended and the proof system is shown to have these properties. It is argued that a large class of informal analyses can be viewed as informal proofs in the proof system.

The development has been performed for a simple language of while programs. It can be generalized to languages with more interesting constructs as for instance recursive procedures with parameters [13]. Also there are alternative strategies for how to extend a total correctness proof system to prove properties of the computation time. One of these strategies is to reason about the time 'used so far' and another is to reason about the time 'required for the rest of the computation' [12, 13]. Although these strategies lead to proof systems satisfying the same sort of soundness and completeness properties as the proof system presented above it seems that the present proof system is the more natural one from the point of view of formalizing informal analyses.

Acknowledgment

I wish to thank John Tucker for his continued interest in my work. I thank the referees for their comments and in particular the suggestion to concentrate on only one of the (originally three) proof systems. Personal thanks are due to Flemming Nielson.

References

[1] P. Aczel, A note on program verification, Manchester University, 1982.
[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms* (Addison-Wesley, Reading, MA, 1982).
[3] K.R. Apt, Ten years of Hoare's logic: a survey—part I, *ACM Toplas* **3** (1981) 431-483.
[4] P.R.J. Asfeld and J.V. Tucker, Complexity theory and the operational structure of algebraic programming systems, *Acta Informat.* **17** (1982) 451-476.
[5] D. Clement, J. Despeyroux, T. Despeyroux and G. Kahn, A simple applicative language: Mini-ML, *Proc. 1986 ACM Conference on Lisp and Functional Programming* (1986).
[6] S.A. Cook, Soundness and completeness of an axiom system for program verification, *SIAM J. Comput.* **7** (1978) 70-90.
[7] D. Harel, *First Order Dynamic Logic*, Lecture Notes in Computer Science **68** (Springer, Berlin, 1979).
[8] C.B. Jones, *Software Development: A Rigorous Approach* (Prentice-Hall, Englewood Cliffs, NJ, 1980).
[9] D.E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (Addison-Wesley, Reading, MA, 1973).
[10] Z. Manna and A. Pnueli, Axiomatic approach to total correctness of programs, *Acta Informat.* **3** (1974) 253-263.
[11] D. Le Metayer, Mechanical analysis of program complexity, *Proc. ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments* (1985).
[12] H.R. Nielson, Proof systems for computation time, *Proc. 3rd Conference on Software Technology and Theoretical Computer Science* (1983).
[13] H.R. Nielson, Hoare logic's for run-time analysis of programs, Ph.D.-thesis, Edinburgh University, 1984.
[14] G.D. Plotkin, A structural approach to operational semantics, Report FN-19, Aarhus University, 1981.
[15] J.R. Shoenfield, *Mathematical Logic* (Addison-Wesley, Reading, MA, 1967).
[16] B. Wegbreit, Verifying program performance, *J. ACM* **23** (1976) 691-699.

Appendix A. Proof of Theorem 4.1

The *soundness result* states that for any well-formed formula $P\langle c:R\rangle Q/V$,

$$\boldsymbol{R} \vdash P\langle c:R\rangle Q/V \text{ implies } \models P\langle c:R\rangle Q/V.$$

It is sufficient to prove that the axioms of $\boldsymbol{R}$ are valid and that the rules preserve validity since then the result follows by induction on the structure of the proofs in $\boldsymbol{R}$.

Case [$\boldsymbol{R}$-1]: To prove

$$\models P\langle x := e:\mathbf{time} = T[e]\rangle I_{V-\{x\}} \& x = \tilde{e}/V$$

assume that $\models P(s)$ for some s. From [$\boldsymbol{S}_t$-1] we get

$$\langle x := e,s\rangle \xrightarrow{e^+(s)} s[e(s)/x].$$

We have $\models I_{V-\{x\}}(s,s[e(s)/x])$ (Fact 10) and also $\models x = \tilde{e}(s,s[e(s)/x])$ (Fact 8). Using the time expressiveness condition we get $\models \mathbf{time} = T[e](s,e^+(s))$. This proves the result.

Case [$\boldsymbol{R}$-2]: Assume

$$\models P \& b\langle c_1:R\rangle Q/V, \tag{A.1}$$

and

$$\models P \& \neg b\langle c_2:R\rangle Q/V. \tag{A.2}$$

To prove

$$\models P\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2:(\mathbf{time} = T[b]) \oplus R\rangle Q/V$$

assume that $\models P(s)$ for some s. If furthermore $\models b(s)$, then (A.1) gives

$$\langle c_1,s\rangle \xrightarrow{t} s', \qquad \models Q(s,s') \quad \text{and} \quad \models R(s,t)$$

for some s' and t. Then [$\boldsymbol{S}_t$-2] gives

$$\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2,s\rangle \xrightarrow{b^+(s)+t} s'.$$

Using the time expressiveness condition we get $\models \mathbf{time} = T[b](s,b^+(s))$ and then Fact 14 gives $\models (\mathbf{time} = T[b]) \oplus R(s,b^+(s)+t)$. This proves the result in the case $\models b(s)$; the case where $\models \neg b(s)$ is similar.

Case [$\boldsymbol{R}$-3]: Assume

$$\models P\langle c_1:R_1\rangle P' \& Q_1/V, \tag{A.3}$$

and

$$\models P'\langle c_2:R_2\rangle Q_2/V. \tag{A.4}$$

To prove

$$\models P\langle c_1;c_2:R_1 \oplus (Q_1 \cdot R_2)\rangle Q_1 \cdot Q_2/V$$

assume that $\models P(s)$ for some s. Then (A.3) gives

$$\langle c_1,s\rangle \xrightarrow{t} s', \qquad \models P'\&Q_1(s,s') \quad \text{and} \quad \models R_1(s,t)$$

for some s' and t. Since P' is pure we have $\models P'(s')$ (Fact 7) so (A.4) gives

$$\langle c_2,s'\rangle \xrightarrow{t'} s'', \qquad \models Q_2(s',s'') \quad \text{and} \quad \models R_2(s',t')$$

for some s'' and t'. From [$\boldsymbol{S}_t$-4] we then get

$$\langle c_1;c_2,s\rangle \xrightarrow{t+t'} s''.$$

From Fact 9 we get $\models Q_1 \cdot Q_2(s,s'')$. Using Fact 13 we get $\models Q_1 \cdot R_2(s,t')$ and then Fact 14 gives $\models R_1 \oplus (Q_1 \cdot R_2)(s,t+t')$. This completes the proof.

Case [$\boldsymbol{R}$-4]: Assume that

$$\models P(z+1)\&b\langle c:R'\rangle P(z)\&Q'/V\cup\{z\}, \tag{A.5}$$

$$\models P(0) \rightarrow \neg b, \tag{A.6}$$

$$\models P(z)\&\neg b\&I_V \rightarrow Q, \tag{A.7}$$

$$\models Q' \cdot Q \rightarrow Q, \tag{A.8}$$

$$\models P(z)\&\neg b\&(\mathbf{time} = T[b]) \rightarrow R, \tag{A.9}$$

$$\models (\mathbf{time} = T[b]) \oplus R' \oplus (Q' \cdot R) \rightarrow R \tag{A.10}$$

where z (not in V) is a program variable ranging over the natural numbers. To prove

$$\models \exists z.P(z)\langle \text{WHILE } b \text{ DO } c:R\rangle Q/V$$

it is sufficient to prove that for any natural number n and state s with $z(s) = n$:

$$\begin{aligned} &\text{if } \models P(z)(s) \text{ then } \langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{t} s', \models Q(s,s') \\ &\qquad\qquad \text{and } \models R(s,t) \text{ for some } s' \text{ and } t. \end{aligned} \tag{A.11}$$

The proof of (A.11) will be by induction on n.

If $n = 0$, then $\models P(0)(s)$ and (A.6) gives $\models \neg b(s)$. Thus [$\boldsymbol{S}_t$-6] gives

$$\langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{b^+(s)} s.$$

Clearly $\models P(z)\&\neg b\&I_V(s,s)$ (Facts 7 and 10) and (A.7) gives $\models Q(s,s)$. The time expressiveness condition gives $\models (\mathbf{time} = T[b])(s,b^+(s))$ and thereby $\models P(z)\&\neg b\&(\mathbf{time} = T[b])(s,b^+(s))$ (Fact 12). The assumption (A.9) then gives $\models R(s,b^+(s))$ proving (A.11) in the base case.

For the induction step assume that (A.11) holds for n and assume that $z(s) = n+1$ and $\models P(z)(s)$. If $\models \neg b(s)$ we proceed as in the case $n = 0$ above so assume that $\models b(s)$. Since z is not in $FV(b)$ ($\subseteq V$ because of the well-formedness condition) we get $\models P(z+1)\&b(s[n/z])$ using Fact 5. Then (A.5) gives

$$\langle c,s[n/z]\rangle \xrightarrow{t} s', \qquad \models P(z)\&Q'(s[n/z],s') \quad \text{and} \quad \models R'(s[n/z],t)$$

for some s' and t. Since $P(z)$ is pure we get $\models P(z)(s')$ (Fact 7). Also z does not occur in $FV(c)$ ($\subseteq V$ because of the well-formedness condition) so $z(s') = n$ follows from Fact 3. Thus the induction hypothesis (A.11) can be applied and we get

$$\langle \text{WHILE } b \text{ DO } c, s' \rangle \xrightarrow{t'} s'', \qquad \models Q(s', s'') \quad \text{and} \quad \models R(s', t')$$

for some s'' and t'. Then [$\boldsymbol{S}_t$-5] gives

$$\langle \text{WHILE } b \text{ DO } c, s[n/z] \rangle \xrightarrow{b^+(s[n/z])+t+t'} s''$$

and since z does not occur in $FV(\text{WHILE } b \text{ DO } c)$ we can replace $s[n/z]$ by s (Fact 4):

$$\langle \text{WHILE } b \text{ DO } c, s \rangle \xrightarrow{b^+(s[n/z])+t+t'} s_0''$$

where $s_0'' \equiv_V s''$. Fact 9 gives $\models Q' \cdot Q(s[n/z], s'')$ and using (A.8) we get $\models Q(s[n/z], s'')$. The well-formedness assumption gives $FV(Q) \subseteq V \cup \tilde{V}$ so $\models Q(s, s_0'')$ follows from Fact 6. The time expressiveness condition gives $\models (\textbf{time} = T[b])(s[n/z], b^+(s[n/z]))$. Fact 13 gives $\models Q' \cdot R(s[n/z], t')$ so using Fact 14 we get $\models (\textbf{time} = T[b]) \oplus R' \oplus (Q' \cdot R)(s[n/z], b^+(s[n/z]) + t + t')$. The assumption (A.10) gives $\models R(s[n/z], b^+(s[n/z]) + t + t')$. The well-formedness condition gives $FV(R) \subseteq V \cup \{\textbf{time}\}$ and using Fact 11 we therefore get $\models R(s, b^+(s[n/z]) + t + t')$. This completes the proof of (A.11).

Case [$\boldsymbol{R}$-5]: Straightforward.

Case [$\boldsymbol{R}$-6]: Straightforward using the Facts 8 and 12.

This completes the proof of the soundness result for $\boldsymbol{R}$.

The *completeness result* of Theorem 4.1 states that for any well-formed formula $P\langle c:R\rangle Q / V$,

$$\models P\langle c:R\rangle Q / V \text{ implies } \boldsymbol{R} \vdash P\langle c:R\rangle Q / V.$$

As argued in Section 4 it is sufficient to prove Lemmas 4.2–4.4.

Proof of Lemma 4.2. The proof is by structural induction on c.

Case $x := e$: Define

$$G_V(x := e): I_{V-\{x\}} \& x = \tilde{e},$$

$$P_V(x := e): \text{TRUE},$$

$$E_V(x := e): \textbf{time} = T[e].$$

First assume that $\langle x := e, s \rangle \xrightarrow{t} s'$. Then [$\boldsymbol{S}_t$-1] and Fact 2 give $t = e^+(s)$ and $s' = s[e(s)/x]$. It is easy to see that $\models G_V(x := e)(s, s')$ (Facts 8 and 10), $\models P_V(x := e)(s)$ and $\models E_V(x := e)(s, t)$ (the time expressiveness condition). Next assume that

$\models G_V(x := e)(s,s')$, i.e. that $s \equiv_{V-\{x\}} s'$ and $e(s) = x(s')$ (Facts 8 and 10). From $[\boldsymbol{S}_t\text{-}1]$ we have $\langle x := e,s\rangle \xrightarrow{e^+(s)} s[e(s)/x]$ and it is easy to see that $s' \equiv_V s[e(s)/x]$. The case where $\models P_V(x := e)(s)$ is trivial using $[\boldsymbol{S}_t\text{-}1]$. So assume that $\models E_V(x := e)(s,t)$. Then $t = e^+(s)$ follows from the time expressiveness condition and the result follows from $[\boldsymbol{S}_t\text{-}1]$.

Case IF b THEN c_1 ELSE c_2: Define

$$G_V(\text{IF}\ldots): (\tilde{b}\&G_V(c_1)) \vee (\neg\tilde{b}\&G_V(c_2)),$$

$$P_V(\text{IF}\ldots): (b\&P_V(c_1)) \vee (\neg b\&P_V(c_2)),$$

$$E_V(\text{IF}\ldots): (\textbf{time} = T[b]) \oplus ((b\&E_V(c_1)) \vee (\neg b\&E_V(c_2))).$$

Assume now that the conditions of the lemma hold for $G_V(c_i)$, $P_V(c_i)$ and $E_V(c_i)$. It is straightforward to prove that they also hold for $G_V(\text{IF}\ldots)$, $P_V(\text{IF}\ldots)$ and $E_V(\text{IF}\ldots)$ using the rules $[\boldsymbol{S}_t\text{-}2]$ and $[\boldsymbol{S}_t\text{-}3]$, the Facts 8, 12 and 14 and the time expressiveness condition.

Case $c_1;c_2$: Define

$$G_V(c_1;c_2): G_V(c_1) \cdot G_V(c_2),$$

$$P_V(c_1;c_2): P_V(c_1)\&\exists X'.(P_V(c_2)\&G_V(c_1))[X'/X][X/\tilde{X}]$$

where X and $\tilde{X}$ are vectors over V and $\tilde{V}$ resp. and X' is a vector of distinct new variables. Furthermore

$$E_V(c_1;c_2): E_V(c_1) \oplus (G_V(c_1) \cdot E_V(c_2)).$$

First assume that $\langle c_1;c_2,s\rangle \xrightarrow{t} s'$. Then $[\boldsymbol{S}_t\text{-}4]$ gives that $t = t' + t''$ and $\langle c_1,s\rangle \xrightarrow{t'} s''$ and $\langle c_2,s''\rangle \xrightarrow{t''} s'$ for some t', t'' and s''. We can then use the induction hypothesis. Fact 9 gives $\models G_V(c_1;c_2)(s,s')$. It is easy to verify that $\models P_V(c_1;c_2)(s)$. Facts 13 and 14 give $\models E_V(c_1;c_2)(s,t)$.

Next assume that $\models G_V(c_1;c_2)(s,s')$. Then Fact 9 gives that for some s'', $\models G_V(c_1)(s,s'')$ and $\models G_V(c_2)(s'',s')$. So using the induction hypothesis we get $\langle c_1,s\rangle \xrightarrow{t} s_0''$ where $s_0'' \equiv_V s''$ and $\langle c_2,s''\rangle \xrightarrow{t'} s_0'$ where $s_0' \equiv_V s'$. Since $FV(c_2) \subseteq V$, Fact 4 gives $\langle c_2,s_0''\rangle \xrightarrow{t'} s_1'$ where $s_1' \equiv_V s_0'$. But then $[\boldsymbol{S}_t\text{-}4]$ gives $\langle c_1;c_2,s\rangle \xrightarrow{t+t'} s_1'$ and $s_1' \equiv_V s'$ as required. Next assume that $\models P_V(c_1;c_2)(s)$. Then $\models P_V(c_1)(s)$ and for some s', $\models P_V(c_2)(s')$ and $\models G_V(c_2)(s,s')$. Because the language is deterministic (Fact 2) this means that $\langle c_1,s\rangle \xrightarrow{t} s_0'$ where $s_0' \equiv_V s'$. Furthermore $\langle c_2,s'\rangle \xrightarrow{t'} s''$ for some s''. From Fact 4 we get $\langle c_2,s_0'\rangle \xrightarrow{t'} s_0''$ for some s_0'' so using $[\boldsymbol{S}_t\text{-}4]$ we get $\langle c_1;c_2,s\rangle \xrightarrow{t+t'} s_0''$ proving the result. Finally, assume that $\models E_V(c_1;c_2)(s,t)$. Then Facts 13 and 14 give that $t = t' + t''$ and $\models E_V(c_1)(s,t')$, $\models G_V(c_1)(s,s')$ and $\models E_V(c_2)(s',t'')$ for some t', t'' and s'. The induction hypothesis gives $\langle c_1,s\rangle \xrightarrow{t'} s_0'$, $\langle c_1,s\rangle \xrightarrow{t_0'} s_1'$ where $s_1' \equiv_V s'$, and $\langle c_2,s'\rangle \xrightarrow{t''} s''$ for some s_0', t_0', s_1' and s''. Fact 2 gives $t' = t_0'$ and $s_0' = s_1'$ and because of Fact 4 and $FV(c_2) \subseteq V$ we get $\langle c_2,s_0'\rangle \xrightarrow{t''} s_0''$ for some s_0''. Then $[\boldsymbol{S}_t\text{-}4]$ gives $\langle c_1;c_2,s\rangle \xrightarrow{t} s_0''$ proving the result.

Case WHILE b DO c: Here we shall make use of the ability to encode finite sequences of elements as a single element. If the body of the loop is executed k times, then this gives rise to $k+1$ states $s_0, s_1, \ldots, s_k$ and $2 * k+1$ natural numbers $t_1, t_2, \ldots, t_{2*k+1}$ where

$$\text{for } 0 \leqslant i < k: \quad \models b(s_i), t_{2*i+1} = b^+(s_i) \text{ and } \langle c, s_i \rangle \xrightarrow{t'} s_{i+1} \text{ where } t' = t_{2*i+2},$$

$$\text{for } i = k: \quad \models \neg b(s_k) \text{ and } t_{2*k+1} = b^+(s_k).$$

Informally we define

$$\begin{aligned} &G_V(\text{WHILE } b \text{ DO } c)(s,s') = \\ &\quad \exists k \exists s_0, \ldots, s_k.(\forall i.0 \leqslant i < k \rightarrow (\tilde{b} \,\&\, G_V(c))(s_i, s_{i+1})) \\ &\qquad\qquad \&\ (\neg b(s_k)) \ \&\ s = s_0 \ \&\ s' = s_k \end{aligned}$$

$$\begin{aligned} &P_V(\text{WHILE } b \text{ DO } c)(s) = \\ &\quad \exists k \exists s_0, \ldots, s_k.(\forall i.0 \leqslant i < k \rightarrow (\tilde{b} \,\&\, G_V(c))(s_i, s_{i+1})) \\ &\qquad\qquad \&\ (\neg b(s_k)) \ \&\ s = s_0. \end{aligned}$$

Below t'_i denotes the sum $\sum_{j=1}^{i} t_j$, that is, t'_{2*i} is the total time required for the first i unfoldings of the loop. Then

$$\begin{aligned} &E_V(\text{WHILE } b \text{ DO } c)(s,t) = \\ &\quad \exists k \exists s_0, \ldots, s_k \exists t'_0, \ldots, t'_{2*k+1}. \\ &\qquad (\forall i.0 \leqslant i < k \rightarrow (\tilde{b} \,\&\, G_V(c))(s_i, s_{i+1})) \\ &\qquad \&\ T[b](s_i) = t'_{2*i+1} - t'_{2*i} \ \&\ E_V(c)(s_i, t'_{2*i+2} - t_{2*i+1}) \\ &\qquad \&\ T[b](s_k) = t'_{2*k+1} - t'_{2*k} \ \&\ (\neg b(s_k)) \\ &\qquad \&\ s = s_0 \ \&\ t'_0 = 0 \ \&\ t'_{2*k+1} = t. \end{aligned}$$

Formally, these formulas are defined using the special formula ϕ introduced in Section 4. Let $V = \{x_1, \ldots, x_m\}$ and define Y to be a vector of distinct new variables $y_1, \ldots, y_m$. Then we define ϕ_V to encode the V-part of the states $s_0, s_1, \ldots, s_k$ as a single vector of elements $u_1, \ldots, u_m$. The free variables of ϕ_V are $V \cup \{y_1, \ldots, y_m, i\}$ and for any natural number n it satisfies

$$\models \phi_V(s[n/i][u_1/y_1] \ldots [u_m/y_m]) \text{ if and only if } s \equiv_V s_n.$$

ϕ_V is defined to be $\phi[x_1/x][y_1/y] \& \cdots \& \phi[x_m/x][y_m/y]$.

We can then define

$$\begin{aligned} &G_V(\text{WHILE } b \text{ DO } c): \\ &\quad \exists k \exists Y.(\forall i \forall X' \forall X''.0 \leqslant i < k \,\&\, \phi_V[X'/X] \ \&\ \phi_V[X''/X][i+1/i] \\ &\qquad\qquad \rightarrow (\tilde{b} \ \&\ G_V(c))[X'/\tilde{X}][X''/X]) \\ &\qquad\quad \&\ (\forall X'.\phi_V[X'/X][k/i] \rightarrow \neg b[X'/X]) \\ &\qquad\quad \&\ \phi_V[\tilde{X}/X][0/i] \ \&\ \phi_V[k/i], \end{aligned}$$

$$
\begin{aligned}
&P_V(\text{WHILE } b \text{ DO } c):\\
&\quad \exists k \exists Y.(\forall i \forall X' \forall X''.0 \leq i < k \ \&\ \phi_V[X'/X] \ \&\ \phi_V[X''/X][i+1/i]\\
&\qquad \rightarrow (\tilde{b}\ \&\ G_V(c))[X'/\tilde{X}][X''/X])\\
&\qquad \&\ (\forall X'.\phi_V[X'/X][k/i] \rightarrow \neg b[X'/X])\\
&\qquad \&\ \phi_V[0/i],
\end{aligned}
$$

$$
\begin{aligned}
&E_V(\text{WHILE } b \text{ DO } c):\\
&\quad \exists k \exists Y \exists y.(\forall i \forall X' \forall X'' \forall t_0 \forall t_1 \forall t_2.\\
&\qquad 0 \leq i < k \ \&\ \phi_V[X'/X] \ \&\ \phi_V[X''/X][i+1/i]\\
&\qquad\quad \&\ \phi[t_0/x][2*i/i]\\
&\qquad\quad \&\ \phi[t_1/x][2*i+1/i] \ \&\ \phi[t_2/x][2*i+2/i]\\
&\qquad \rightarrow (\tilde{b}\ \&\ G_V(c))[X'/\tilde{X}][X''/X] \ \&\ T[b][X'/X] = t_1 - t_0\\
&\qquad\quad \&\ E_V(c)[X'/X][t_2 - t_1/\mathbf{time}])\\
&\qquad \&\ (\forall X' \forall t_0 \forall t_1.\\
&\qquad\quad \phi_V[X'/X][k/i] \ \&\ \phi[t_0/x][2*k/i] \ \&\ \phi[t_1/x][2*k+1/i]\\
&\qquad\quad \rightarrow \neg b[X'/X] \ \&\ T[b][X'/X] = t_1 - t_0 \ \&\ \mathbf{time} = t_1)\\
&\qquad \&\ \phi_V[0/i] \ \&\ \phi[0/x][0/i]
\end{aligned}
$$

It is fairly straightforward to see that these formulas express the required relationships between the states $s_0, s_1, \ldots, s_k$ and the accumulated computation times $t'_0, t'_1, \ldots, t'_{2*k+1}$. The actual proof showing that the formulas fulfill the conditions of the lemma is by induction on k, the number of unfoldings of the loop. We shall omit the rather tedious proof. □

Proof of Lemma 4.3. To prove the lemma assume that $\models P\langle c{:}R\rangle Q/V$. It is easy to see that $\models P \rightarrow P_V(c)$. To show $\models \tilde{P}\&G_V(c) \rightarrow Q$ assume that $\models \tilde{P}\&G_V(c)(s,s')$ for some s and s'. Then $\models P(s)$ (Fact 8), so

$$\langle c,s\rangle \xrightarrow{t'} s'', \qquad \models Q(s,s'') \quad \text{and} \quad \models R(s,t') \tag{A.12}$$

for some s'' and t'. From $\models G_V(c)(s,s')$ we get $\langle c,s\rangle \xrightarrow{t''} s'_0$ for some t'' and s'_0 with $s' \equiv_V s'_0$. The language is deterministic (Fact 2), so $s'' = s'_0$. The well-formedness assumption gives $FV(Q) \subseteq V \cup \tilde{V}$, so $\models Q(s,s')$ follows from Fact 6 and $s' \equiv_V s''$. To show $\models P\&E_V(c) \rightarrow R$ assume $\models P\&E_V(c)(s,t)$. Then $\models P(s)$ (Fact 12) and (A.12) holds. From $\models E_V(c)(s,t)$ we have $\langle c,s\rangle \xrightarrow{t} s'$ for some s' and since the language is deterministic (Fact 2) we get $t = t'$ and thereby $\models R(s,t)$. □

Proof of Lemma 4.4. This lemma is shown by structural induction on c.

Case $x := e$: Using [***R***-1] we get

$$\boldsymbol{R} \vdash P_V(x := e)\langle x := e{:}\mathbf{time} = T[e]\rangle I_{V-\{x\}}\&x = \tilde{e}/V$$

and since $E_V(x := e)$ is $\mathbf{time} = T[e]$ and $G_V(x := e)$ is $I_{V-\{x\}}\&x = \tilde{e}$ (see the proof of Lemma 4.2) this completes the proof.

Case IF b THEN c_1 ELSE c_2: By assumption we have the proofs

$$\boldsymbol{R} \vdash P_V(c_i)\langle c_i : E_V(c_i)\rangle G_V(c_i)/V$$

for $i = 1, 2$. Using the definitions of $P_V(\text{IF}\ldots)$ and $G_V(\text{IF}\ldots)$ in the proof of Lemma 4.2 it is easy to verify that

$$\begin{aligned}
&\models P_V(\text{IF}\ldots)\&b \rightarrow P_V(c_1),\\
&\models P_V(\text{IF}\ldots)\&b\&E_V(c_1) \rightarrow (b\&E_V(c_1)) \vee (\neg b\&E_V(c_2)),\\
&\models P_V(\text{IF}\ldots)\&\tilde{b}\,\&G_V(c_1) \rightarrow G_V(\text{IF}\ldots),
\end{aligned}$$

so using the rules [$\boldsymbol{R}$-5] and [$\boldsymbol{R}$-6] we get

$$\boldsymbol{R} \vdash P_V(\text{IF}\ldots)\,\&b\langle c_1 : ((b\&E_V(c_1)) \vee (\neg b\&E_V(c_2)))\rangle G_V(\text{IF}\ldots)/V$$

and similarly

$$\boldsymbol{R} \vdash P_V(\text{IF}\ldots)\&\neg b\langle c_2 : ((b\&E_V(c_1)) \vee (\neg b\&E_V(c_2)))\rangle G_V(\text{IF}\ldots)/V.$$

So using [$\boldsymbol{R}$-2] we get

$$\boldsymbol{R} \vdash P_V(\text{IF}\ldots)\langle \text{IF}\ldots : E_V(\text{IF}\ldots)\rangle G_V(\text{IF}\ldots)/V$$

because $E_V(\text{IF}\ldots)$ is defined to be $(\mathbf{time} = T[b]) \oplus ((b\&E_V(c_1)) \vee (\neg b\&E_V(c_2)))$ (see the proof of Lemma 4.2).

Case $c_1;c_2$: By assumption we have

$$\boldsymbol{R} \vdash P_V(c_i)\langle c_i : E_V(c_i)\rangle G_V(c_i)/V$$

for $i = 1,2$. It is easy to verify that

$$\models P_V(c_1;c_2) \rightarrow P_V(c_1)$$

using the definitions in the proof of Lemma 4.2. Using Fact 2 and [$\boldsymbol{S}_t$-4] it is easy to show that

$$\models P_V(c_1;c_2)\&G_V(c_1) \rightarrow P_V(c_2)\&G_V(c_1),$$

so using the rules [$\boldsymbol{R}$-5] and [$\boldsymbol{R}$-6] we get

$$\boldsymbol{R} \vdash P_V(c_1;c_2)\langle c_1 : E_V(c_1)\rangle P_V(c_2)\&G_V(c_1)/V.$$

Using [$\boldsymbol{R}$-3] we get

$$\boldsymbol{R} \vdash P_V(c_1;c_2)\langle c_1;c_2 : E_V(c_1;c_2)\rangle G_V(c_1;c_2)/V$$

since, using the definitions in the proof of Lemma 4.2, $G_V(c_1;c_2)$ is $G_V(c_1) \cdot G_V(c_2)$ and $E_V(c_1;c_2)$ is $E_V(c_1) \oplus (G_V(c_1) \cdot E_V(c_2))$.

Case WHILE b DO c: In order to define the invariant $P'(z)$ we shall transform c into $c' = \text{IF}\ z = 0$ THEN *loop* ELSE $(c; z := z - 1)$ where *loop* is a program that never

terminates. Then $P'(z)$ can be defined to be $P_{V\cup\{z\}}$(WHILE b DO c'). The relationship between c and c' is semantically expressed by

$$\begin{array}{l}\text{for any state } s \text{ with } z(s)>0\\ \quad \langle c,s\rangle \xrightarrow{t} s' \text{ if and only if } \langle c',s\rangle \xrightarrow{t'} s''\\ \text{where } s'' \equiv_V s' \text{ and } z(s'')=z(s)-1.\end{array} \tag{A.13}$$

It is not hard to show that

$$\begin{array}{l}\text{for any state } s \text{ there exists a natural number } n \text{ such that}\\ \quad \langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{t} s'\\ \text{if and only if}\\ \quad \langle \text{WHILE } b \text{ DO } c',s[n/z]\rangle \xrightarrow{t'} s''.\end{array} \tag{A.14}$$

The induction hypothesis gives us a proof

$$\boldsymbol{R} \vdash P_{V\cup\{z\}}(c)\langle c{:}E_{V\cup\{z\}}(c)\rangle G_{V\cup\{z\}}(c)/V\cup\{z\}.$$

Below we shall prove that

$$\models P'(z+1)\&b \rightarrow P_{V\cup\{z\}}(c), \tag{A.15}$$

and

$$\models P'(z+1)\&\tilde{b}\,\&G_{V\cup\{z\}}(c) \rightarrow P'(z)\&\tilde{b}\,\&G_{V\cup\{z\}}(c), \tag{A.16}$$

so using [$\boldsymbol{R}$-5] and [$\boldsymbol{R}$-6] we get

$$\boldsymbol{R} \vdash P'(z+1)\&b\langle c{:}b\&E_{V\cup\{z\}}(c)\rangle P'(z)\&\tilde{b}\,\&G_{V\cup\{z\}}(c)/V\cup\{z\}.$$

Below we show

$$\models P'(0) \rightarrow \neg b, \tag{A.17}$$

$$\models P'(z)\&\neg b\&I_V \rightarrow G_V(\text{WHILE } b \text{ DO } c), \tag{A.18}$$

$$\models (\tilde{b}\,\&G_{V\cup\{z\}}(c))\cdot G_V(\text{WHILE } b \text{ DO } c) \rightarrow G_V(\text{WHILE } b \text{ DO } c), \tag{A.19}$$

$$\models P'(z)\&\neg b\&(\textbf{time}=T[b]) \rightarrow E_V(\text{WHILE } b \text{ DO } c), \tag{A.20}$$

$$\begin{array}{l}\models (\textbf{time}=T[b])\oplus(b\&E_{V\cup\{z\}}(c))\\ \quad \oplus((\tilde{b}\,\&G_{V\cup\{z\}}(c))\cdot E_V(\text{WHILE } b \text{ DO } c))\\ \quad \rightarrow E_V(\text{WHILE } b \text{ DO } c).\end{array} \tag{A.21}$$

Thus [$\boldsymbol{R}$-4] can be applied and gives

$$\begin{array}{l}\boldsymbol{R} \vdash \exists z.\, P'(z)\langle \text{WHILE } b \text{ DO } c{:}E_V(\text{WHILE } b \text{ DO } c)\rangle\\ \qquad\qquad G_V(\text{WHILE } b \text{ DO } c)/V.\end{array}$$

Using [$\boldsymbol{R}$-5] with

$$\models P_V(\text{WHILE } b \text{ DO } c) \rightarrow \exists z'.\, P'(z) \tag{A.22}$$

which is shown below we get

$$\boldsymbol{R} \vdash P_V(\text{WHILE } b \text{ DO } c)\langle \text{WHILE } b \text{ DO } c : E_V(\text{WHILE } b \text{ DO } c)\rangle \\ G_V(\text{WHILE } b \text{ DO } c)/V$$

as required.

The truth of the formulas (A.15)-(A.22) is shown using the relationships (A.13) and (A.14) as well as the characterization of the formulas $P_{\ldots}(\ldots)$, $G_{\ldots}(\ldots)$ and $E_{\ldots}(\ldots)$.

To prove (A.15) assume that $\models P'(z+1)\&b(s)$. Then $\models P'(z)(s[n/z])$ where $n = z(s)+1$ and Lemma 4.2 gives $\langle \text{WHILE } b \text{ DO } c', s[n/z]\rangle \xrightarrow{t} s''$ for some t and s''. Since $FV(b) \subseteq V$ we have $\models b(s[n/z])$ (Fact 5) and $[\boldsymbol{S}_t\text{-}5]$ gives $\langle c', s[n/z]\rangle \xrightarrow{t'} s_0'$ and $\langle \text{WHILE } b \text{ DO } c', s_0'\rangle \xrightarrow{t''} s''$. Using (A.13) we get $\langle c, s[n/z]\rangle \xrightarrow{t_0'} s_1'$ where $s_0' \equiv_V s_1'$ and $z(s_0') = n-1$. Since $FV(c) \subseteq V$ we can use Fact 4 and get $\langle c, s\rangle \xrightarrow{t_0'} s_2'$ for some s_2' with $s_2' \equiv_V s_1'$. Lemma 4.2 gives that $\models P_{V\cup\{z\}}(c)(s)$ so (A.15) follows.

To prove (A.16) assume that $\models P'(z+1)\&\tilde{b}\,\&G_{V\cup\{z\}}(c)(s,s')$. Fact 8 gives $\models P'(z+1)\&b(s)$ and as in the proof of (A.15) above we get $\langle c, s\rangle \xrightarrow{t_0'} s_2'$ and $\langle \text{WHILE } b \text{ DO } c', s_0'\rangle \xrightarrow{t''} s''$ where $z(s_0') = z(s)$ and $s_2' \equiv_V s_0'$. Thus $\models P'(z)(s_0')$. From $\models G_{V\cup\{z\}}(c)(s,s')$, Lemma 4.2 and Fact 2 we get $s' \equiv_{V\cup\{z\}} s_2'$ and thereby $s' \equiv_V s_0'$. Since z is not in $FV(c)$ we have $z(s) = z(s_0')$ (Fact 3) so $z(s') = z(s_0')$. From Lemma 4.2 we have $FV(P'(z)) \subseteq V \cup \{z\}$, so using Fact 5 with $s_0' \equiv_{V\cup\{z\}} s'$ we get $\models P'(z)(s')$ and this proves (A.16).

To prove (A.17) assume $\models P'(0)(s)$. Then $\models P'(z)(s[0/z])$, so Lemma 4.2 gives $\langle \text{WHILE } b \text{ DO } c', s[0/z]\rangle \xrightarrow{t} s'$ and from the axioms and rules of $\boldsymbol{S}_t$ it follows that $\models \neg b(s[0/z])$ since otherwise the program *loop* of c' will be executed. Using Fact 5 we get $\models \neg b(s)$ and (A.17) has been proved.

To prove (A.18) assume that $\models P'(z)\&\neg b\&I_V(s,s')$. Then $\models P'(z)\&\neg b(s')$ (Fact 7) and since $FV(b) \subseteq V$ and $s \equiv_V s'$ (Fact 10) we get $\models \neg b(s')$ (Fact 5). Thus $[\boldsymbol{S}_t\text{-}6]$ gives $\langle \text{WHILE } b \text{ DO } c, s\rangle \xrightarrow{t} s$ and $\models G_V(\text{WHILE } b \text{ DO } c)(s,s)$ follows from Lemma 4.2. Since $FV(G_V(\text{WHILE } b \text{ DO } c)) \subseteq V \cup \tilde{V}$ and $s \equiv_V s'$, we get $\models G_V(\text{WHILE } b \text{ DO } c)(s,s')$ as required (Fact 6).

To prove (A.19) assume that $\models (\tilde{b}\,\&G_{V\cup\{z\}}(c)) \cdot G_V(\text{WHILE } b \text{ DO } c)(s,s'')$, that is, for some s', $\models \tilde{b}\,\&G_{V\cup\{z\}}(c)(s,s')$ and $\models G_V(\text{WHILE } b \text{ DO } c)(s',s'')$ (Fact 9). Then $\models b(s)$ (Fact 8) and Lemma 4.2 gives $\langle c, s\rangle \xrightarrow{t} s_0'$ where $s_0' \equiv_V s'$, and $\langle \text{WHILE } b \text{ DO } c, s'\rangle \xrightarrow{t'} s_0''$, where $s_0'' \equiv_V s''$. Since $FV(\text{WHILE } b \text{ DO } c) \subseteq V$, we get using Fact 4 $\langle \text{WHILE } b \text{ DO } c, s_0'\rangle \xrightarrow{t'} s_1''$ where $s_1'' \equiv_V s_0''$. Then $[\boldsymbol{S}_t\text{-}5]$ gives $\langle \text{WHILE } b \text{ DO } c, s\rangle \xrightarrow{t''} s_1''$ and since $s_1'' \equiv_V s''$, we get $\models G_V(\text{WHILE } b \text{ DO } c)(s,s'')$ using Lemma 4.2.

To prove (A.20) assume that $\models P'(z)\&\neg b\&(\textbf{time} = T[b])(s,t)$. Then $\models \neg b(s)$ (Fact 12), so $[\boldsymbol{S}_t\text{-}6]$ gives $\langle \text{WHILE } b \text{ DO } c, s\rangle \xrightarrow{b^+(s)} s$. The time expressiveness condition gives $b^+(s) = T[b](s)$, so $t = b^+(s)$. But then Lemma 4.2 gives $\models E_V(\text{WHILE } b \text{ DO } c)(s,t)$.

To prove (A.21) assume that $\models (\textbf{time} = T[b]) \oplus E_{V\cup\{z\}}(c) \oplus (\tilde{b}\,\&G_{V\cup\{z\}}(c) \cdot E_V(\text{WHILE } b \text{ DO } c))(s,t)$. Then Facts 13 and 14 give that $t = t_0 + t' + t''$

where $t_0 = T[b](s)$, $\models E_{V\cup\{z\}}(c)(s,t')$ and for some s', $\models \tilde{b}\,\&\,G_{V\cup\{z\}}(c)(s,s')$ and $\models E_V(\text{WHILE } b \text{ DO } c)(s',t'')$. Using Lemma 4.2 and Fact 2 we get $\langle c,s\rangle \xrightarrow{t'} s_0'$ where $s_0' \equiv_{V\cup\{z\}} s'$ and $\langle \text{WHILE } b \text{ DO } c,s'\rangle \xrightarrow{t''} s''$. Again $FV(\text{WHILE } b \text{ DO } c) \subseteq V$, so Fact 4 gives $\langle \text{WHILE } b \text{ DO } c,s_0'\rangle \xrightarrow{t''} s_0''$. Since $\models b(s)$ (Fact 8), $[S_t\text{-}5]$ gives $\langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{b^+(s)+t'+t''} s_0''$. The time expressiveness condition gives $b^+(s) = T[b](s)$, so $\models E_V(\text{WHILE } b \text{ DO } c)(s,t)$ follows from Lemma 4.2.

Finally, to prove (A.22) assume that $\models P_V(\text{WHILE } b \text{ DO } c)(s)$. Then Lemma 4.2 gives $\langle \text{WHILE } b \text{ DO } c,s\rangle \xrightarrow{t} s'$ for some s' and t. From (A.14) we get that for some natural number n, $\langle \text{WHILE } b \text{ DO } c',s[n/z]\rangle \xrightarrow{t'} s''$ and then Lemma 4.2 gives $\models P'(z)(s[n/z])$ and thereby $\models \exists z.P'(z)(s)$ as required.

This completes the proof of Lemma 4.4. □